Itertools functions:

- Count():

    - Infinite looping
    - Syntax: count(start point, step)
      Where start point default value is 0 and
      Step default value is 1
    - Eg.
      Count()→0,1,2,3,4,-------infinite
      Count(5)→5,6,7,8,9,-----infinite
      Count(1,2)→1,3,5,7,9---infinite
    - To print this always need iteration printing.
      Eg

```
x = count()
print(x)
for item in x:
   print(item)
```

- Cycle():
    - Infinite looping
    - Syntax: cycle(iterable)
    - Eg. v1 = [10,20,30,40,50]

```
x = cycle(v1)
for item in x:
   print(item)
```

o/p->10,20,30,40,50, 10,20,30,40,50,10,20---infinite

- Repeat():
    - Infinite/finite looping
    - Syntax: repeat(iterable, time)
    - V1=[1,2,3]
      Repeat(v1)
      Infinite looping→[1,2,3], [1,2,3], [1,2,3], [1,2,3]---
    - V1=[1,2,3]
      Repeat(v1,2)
      finite looping→[1,2,3], [1,2,3]

- Accumulate():
    - Syntax: accumulate(iterator, function, initial value)
    - v1 = [8,2,3,9,1,5]
    - result = list(accumulate(v1,(**lambda** x1,x2 : x1+x2),initial=0))
      result = [0, 8, 10, 13, 22, 23, 28]
    - result = list(accumulate(v1,(**lambda** x1,x2 : x1+x2)))
      result = [8, 10, 13, 22, 23, 28]

- chain():
    - used to combine different datatype in single variable to perform operation on it.
    - Syntax: chain(iterable1,iterable2----iterableN)
    - v1 = [8,2,3,9,1,5]
      v2 = (11,22,33,44,55,66)
      z = chain(v1,v2)
      z = 8,2,3,9,1,5, 11,22,33,44,55,66

    - here v1 is list n v2 is tuple

- chain.from_iterable():

    - similar to chain only difference is its take only one iterable.
    - Syntax = chain.from_iterable(iterable)

    -
      ```
      v1 = [[9,10],(1,2,3,4),{'a','b','c'}]
      z = list(chain.from_iterable(v1))
      z = [9, 10, 1, 2, 3, 4, 'b', 'c', 'a']
      ```

- copmpress():

    - it compress the value using selector(predicate i.e True or False)
    - True →any value,True,1
False→False,None,0,empty list/tuple/set/dict
    - Only return where selector is True
    - Eg

    -
      ```
      v1 = ['aa','bb','cc','dd','ee','ff']
      v2 = [1,False,True,None,55,[]]
      z = list(compress(v1,v2))
      z = ['aa', 'cc', 'ee']
      ```

- dropwhile():

    - syntax: dropwhile(predicate fun, iterable)
    - it start collecting when first false found

    -
      ```
      v1 = [98,56,44,52,33,71,12,36,76,42,25]
      z = list(dropwhile(lambda x :x%2==0,v1))
      z = [33, 71, 12, 36, 76, 42, 25]
      ```

- takewhile():
  - syntax: takewhile(predicate fun, iterable)
  - collecting although first condition found false i.e. it is apposite to dropwhile

  -
    ```
    v1 = [98,56,44,52,33,71,12,36,76,42,25]
    z = list(takewhile(lambda x :x%2==0,v1))
    z = [98, 56, 44, 52]
    ```

- filterfalse():
  - syntax: filterfalse(predicate,iterable)
  - apposite to filter

  -
    ```
    v1 = [98,11,56,63,44,52,33,71,12,36,76,42,25]
    z = list(filterfalse(lambda x :x%2==0,v1))
    z= [11, 63, 33, 71, 25]
    ```

- islicing()
  - slicing like list
  - syntax: isslicing(iterable, stop point)
    or isslicing(iterable, start point, stop point)
  - based on indexing start index inclusive or end index exclusive.

  -
    ```
    v1 = [98,11,56,63,44,52,33,71,12,36,76,42,25]
    z = list(islice(v1,5))
    z = [98, 11, 56, 63, 44]
    ```
    where 5 consider as stop point

  -
    ```
    v1 = [98,11,56,63,44,52,33,71,12,36,76,42,25]
    z = list(islice(v1,3,8))
    z= [63, 44, 52, 33, 71]
    ```

- zip_longest():
  - making pair = min len of iterable

  -
    ```
    v1 = [1,2,3,4,5,6]
    v2=[10,20,30,40]

    x1 = list(zip(v1,v2))
    x2 = list(zip_longest(v1,v2))
    →x1 = [(1, 10), (2, 20), (3, 30), (4, 40)]
     X2=[(1, 10), (2, 20), (3, 30), (4, 40), (5, None), (6, None)]
    ```

  -
    ```
    y1 = list(zip(v2,v1))
    y2 = list(zip_longest(v2,v1))
    →y1 = [(10, 1), (20, 2), (30, 3), (40, 4)]
    Y2= [(10, 1), (20, 2), (30, 3), (40, 4), (None, 5), (None, 6)]
    ```

- starmap():
  - used when data is complex of complex
    [(),(),()]
  - v1 = [(1,'a',3),(4,'b',2),(3,'c',2,)]

    using map:for logic we need indexing its get complicated

    ```
    z = list(map(lambda x :x[0]*x[1]*x[2],v1))
    ```

    using starmap:

    ```
    s = list(starmap(lambda x1,x2,x3 :x1*x2*x3,v1))

    o/p→ ['aaa', 'bbbbbbbb', 'cccccc']
    ```

- product():

  - self + forward pair + backward pair
  - syntax: product(iterable, repeat=)
  - a = [1,2,3,4]
    p = list(product(a, repeat=3))
  - p=[(1, 1, 1), (1, 1, 2), (1, 1, 3), (1, 1, 4), (1, 2, 1), (1, 2, 2), (1, 2, 3), (1, 2, 4), (1, 3, 1), (1, 3, 2), (1, 3, 3), (1, 3, 4), (1, 4, 1), (1, 4, 2), (1, 4, 3), (1, 4, 4), (2, 1, 1), (2, 1, 2), (2, 1, 3), (2, 1, 4), (2, 2, 1), (2, 2, 2), (2, 2, 3), (2, 2, 4), (2, 3, 1), (2, 3, 2), (2, 3, 3), (2, 3, 4), (2, 4, 1), (2, 4, 2), (2, 4, 3), (2, 4, 4), (3, 1, 1), (3, 1, 2), (3, 1, 3), (3, 1, 4), (3, 2, 1), (3, 2, 2), (3, 2, 3), (3, 2, 4), (3, 3, 1), (3, 3, 2), (3, 3, 3), (3, 3, 4), (3, 4, 1), (3, 4, 2), (3, 4, 3), (3, 4, 4), (4, 1, 1), (4, 1, 2), (4, 1, 3), (4, 1, 4), (4, 2, 1), (4, 2, 2), (4, 2, 3), (4, 2, 4), (4, 3, 1), (4, 3, 2), (4, 3, 3), (4, 3, 4), (4, 4, 1), (4, 4, 2), (4, 4, 3), (4, 4, 4)]

- permutations():

  - forward pair + backward pair
  - syntax: permutations(iterable, r=)
  - a = [1,2,3,4]
    p1 = list(permutations(a, r=3))
  - p1= [(1, 2, 3), (1, 2, 4), (1, 3, 2), (1, 3, 4), (1, 4, 2), (1, 4, 3), (2, 1, 3), (2, 1, 4), (2, 3, 1), (2, 3, 4), (2, 4, 1), (2, 4, 3), (3, 1, 2), (3, 1, 4), (3, 2, 1), (3, 2, 4), (3, 4, 1), (3, 4, 2), (4, 1, 2), (4, 1, 3), (4, 2, 1), (4, 2, 3), (4, 3, 1), (4, 3, 2)]

- combinations():

  - forward pair
  - syntax: combinations(iterable, r=)
  - a = [1,2,3,4]
    p1 = list(combinations(a, r=3))
  - p1 = [(1, 2, 3), (1, 2, 4), (1, 3, 4), (2, 3, 4)]

- combinations_with_replacement():

  - self + forward pair
  - syntax: combinations_with_replacement(iterable, r=)
  - a = [1,2,3,4]
    p1 = list(combinations_with_replacement(a, r=3))
  - p1 = [(1, 1, 1), (1, 1, 2), (1, 1, 3), (1, 1, 4), (1, 2, 2), (1, 2, 3), (1, 2, 4), (1, 3, 3), (1, 3, 4), (1, 4, 4), (2, 2, 2), (2, 2, 3), (2, 2, 4), (2, 3, 3), (2, 3, 4), (2, 4, 4), (3, 3, 3), (3, 3, 4), (3, 4, 4), (4, 4, 4)]

- groupby()

- pairwise()

- tee()